Using naive Bayes, logistic regression and SVM for handwritten characters classification
T-61.3050 Term Project, final report

Géraud Le Falher GERAUD.LEFALHER@AALTO.FI
Student number **336 978**

Abstract

After looking at the dataset of handwritten characters, I examine the effect of supervised (LDA) and unsupervised (PCA) dimensionality reduction. Then I used supervised classification, either with generative (naive Bayes) or not (logistic regression, SVM) model, which give respectively 66.2%, 69.8% and 89.5% of accuracy.

1. Dataset

The training dataset consists of 42,152 handwritten characters, each represented by a binary matrix with 8 columns and 16 rows. Each 1 maps to a white pixel and each 0 to a black one, thus we can visualize some of them[1] to get a sense of what they may look like, like in Figure 1. We can also look at the frequency of each letter in the Table 1. It shows us that the corpus does not follow the relative frequencies of letters in the English language because for instance, there is more N than E.

After making these observations, I tried to apply advice given in Chapter 11 of Alpaydin's book (Alpaydin, 2009), relative to the preprocessing of characters. But they are already in the center of each picture, and scaled to occupy all the possible space. What could maybe be improved is their rotation but it sounds like a daunting task and apart from that, the dataset is of an excellent quality. So the only thing I did was to convert the text file to MATLAB binary format in order to speed-up subsequents loadings.

[1] I adapt some code of the ml-class courses of Prof. Ng

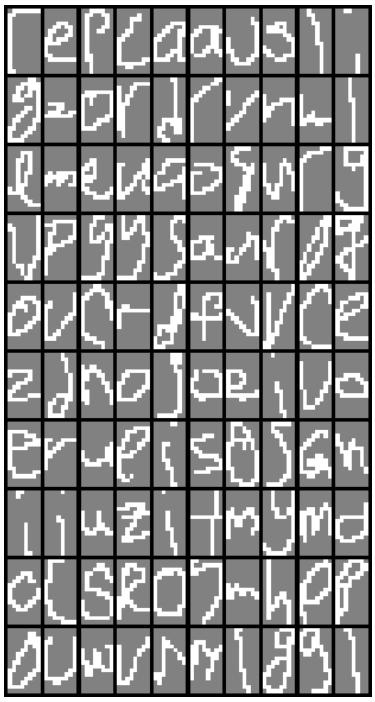


Figure 1: A hundred characters from the training set

2. Dimensionality reduction

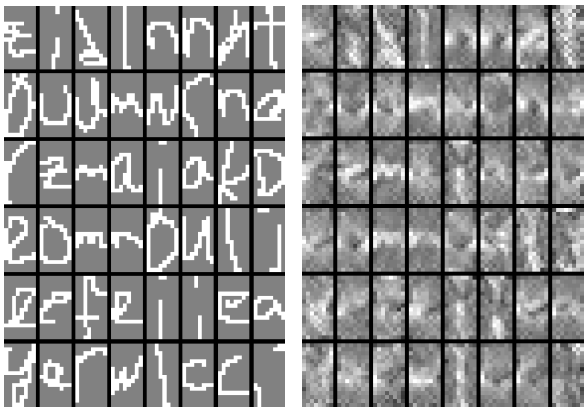
2.1. Linear Discriminant Analysis

I started with linear discriminant analysis because as a supervised method, it seems more promising. For each class, I computed the within class scatter matrix:

$$S_W = \sum_{i=1}^{K} S_i$$

where S_i is the covariance matrix of all the x which belongs to class i and m_i is their mean:

$$S_i = \sum_{i=1}^{K} r_i^t (x^t - m_i)(x^t - m_i)^T$$



(a) The first 48 characters of the training set.

(b) Their reconstruction in a 25 dimension space.

Figure 2: Before/After comparison of LDA reduction

and the between class scatter matrix:

$$S_B = \sum_{i=1}^{K} N_i (m_i - m)(m_i - m)^T$$

where N_i is the size of class i and m the mean of all m_i

Then I get the projection matrix on a subspace of dimension d by taking the first d largest eigenvectors of $S_W^{-1} S_B$ with the constraint that $d < K$, because the rank of S_W is at most K. But the reconstruction was visually blurry (see Figure 2) so I decided to try PCA instead, while later tests show me that it was a wrong assumption.

2.2. Principal Components Analysis

PCA is an unsupervised method which consists of building the projection matrix by taking the first d largest eigenvectors of the covariance matrix S, after having centered the data and normalized the variances, which can be done with the MATLAB function `zscore`. Alpaydin suggests that we can still use the label information by using the sum of the covariance matrix of each class weighted by their estimated probability:

$$S = \sum_{i=1}^{K} \hat{P}(C_i) S_i$$

I did that and it effectively provides a minor improvement in terms of proportion of variance explained, as shown in Figure 3.

A	B	C	D	E	F	G
2958	860	1706	880	3763	723	2239
H	I	J	K	L	M	N
623	3953	189	817	2275	1416	4353
O	P	Q	R	S	T	U
3440	1283	140	2351	1090	1594	2258
	V	W	X	Y	Z	
	664	140	413	1033	991	

Table 1: Letters frequencies in the training dataset

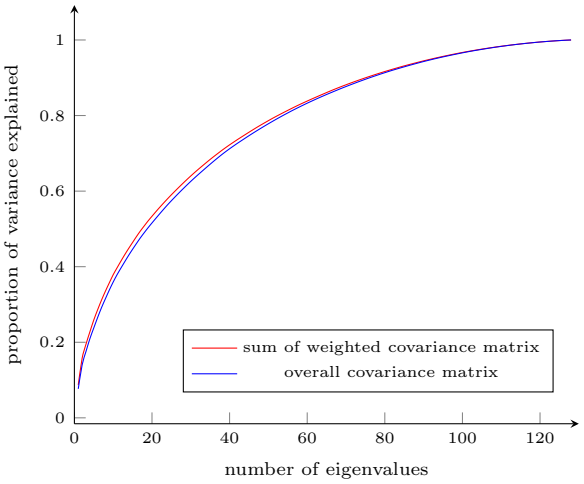


Figure 3: Using label information leads to a (little) more efficient selection of eigenvectors

The reconstruction was also more appealing visually, as shown in Figure 4.

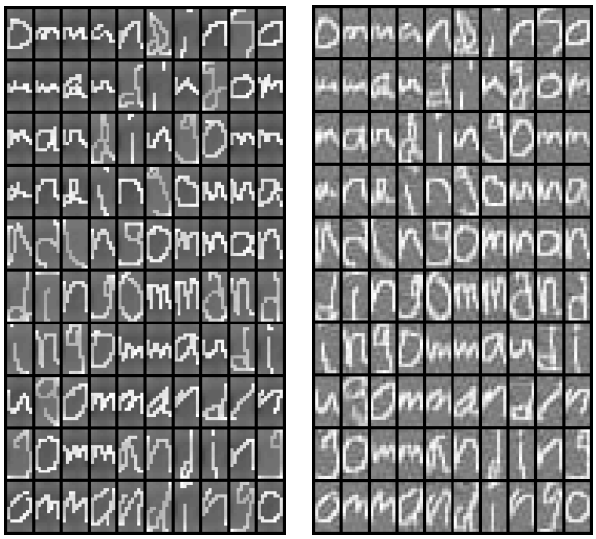
3. Naive Bayes classifier

Because after the dimensionality reduction, each feature seems to approximatively follow a normal distribution (see Figure 5), I decided to start with a naive Bayes classifier. Of course, it requires to verify these assumptions afterward but as it is easy to implement and very fast to train, it sounds like a good baseline.

Basically, we suppose that $p(x|C_i) \sim \mathcal{N}_d(\mu_i, \Sigma_i)$ so the discriminant function for class i is $g_i(x) = \log(p(x|C_i)) + \log(P(C_i))$ and when we plug the usual estimators

$$m_i = \frac{1}{N} \sum_{t=1}^{N} x_i^t \qquad \text{and}$$

$$s_{i,j} = \frac{1}{N} \sum_{t=1}^{N} (x_i^t - m_i)(x_j^t - m_j)^T$$



(a) 100 characters after `zscore`.

(b) Their reconstruction in a 63 dimension space.

Figure 4: Before/After comparison of PCA reduction

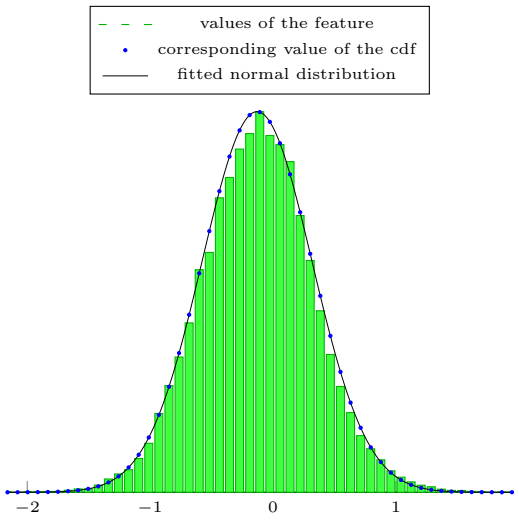


Figure 5: Distribution of the 28$^{\text{th}}$ feature given by PCA.

we get

$$g_i(x) = \mathbf{w_i}^T x + w_{i0} \qquad \text{with}$$
$$\mathbf{w_i} = S^{-1} m_i \qquad \text{and}$$
$$w_{i0} = -\frac{1}{2} m_i^T S^{-1} m_i + \log(\hat{P}(C_i))$$

The result were not very good because this method computes $Kd(d+1)/2 = 54{,}080$ parameters with a dimensionality reduced to 64[2] by PCA whereas there is only 42,152 samples and even less in cross-validation. So I wrote the covariance matrix in the form

$$S_i^* = \alpha\sigma^2\mathbf{I} + \beta S + (1-\alpha-\beta)S_i \qquad \text{with}$$
$$S = \sum_i^K \hat{P}(C_i) S_i$$

and performed a grid search with 4-fold cross validation to find optimal α and β, namely 0 and 1.

To find the best parameter d of PCA reduction, I did a 5-fold cross validation with several values and it turns out that the accuracy reaches a maximum of around 69.1% for $d = 63$ and then becomes somewhat flat (see Figure 6). It appears that this generalization accuracy is rather optimistic because on the test set, the accuracy is only 66.2%.

I earlier mentioned that I discarded LDA whereas it was not as bad as it looks and indeed, we can see in Figure 7 that naive Bayes classifier performs almost as well with only 11 features generated by LDA.

[2]Number chosen because it captures 85% of the variance.

To summarize, while naive Bayes' approach is easy to implement and takes less than a second to be trained and to give a prediction, it does not yield very good results. It is probably because its primary assumption, the "normality" of data, does not hold in a 63-dimensional space with 26 classes. Of course, we could try to use a better generative model in the form of mixture of gaussians but that seems more complicated.

4. Logistic regression

Because doing a full density estimation could be challenging in a high dimensional space and also because it is not required to do the classification, I then tried a linear discrimination classification using the sigmoid function. As seen in exercise session 5, with two class, we could write the log-likelihood as:

$$\mathcal{L} = \sum_{t=1}^{N} \left(r^t \log y^t + (1-r^t)\log(1-y^t) \right) - \lambda \left|\left| \mathbf{w}^T \right|\right|^2$$

with

$$y^t = \text{sigmoid}(\mathbf{w}^T x^t) = \frac{1}{1+\exp(-\mathbf{w}^T x^t)}$$

and where λ is a regularization parameter that ensures that $\mathbf{w}$ does not become too large in order to accommodate for the training set and thus be prone to overfitting.

There is no analytical solution for $\mathbf{w}$ so we need a gradient based method in order to optimize it. Since I found a minimization function on internet[3], I computed

[3]by Carl Edward Rasmussen.

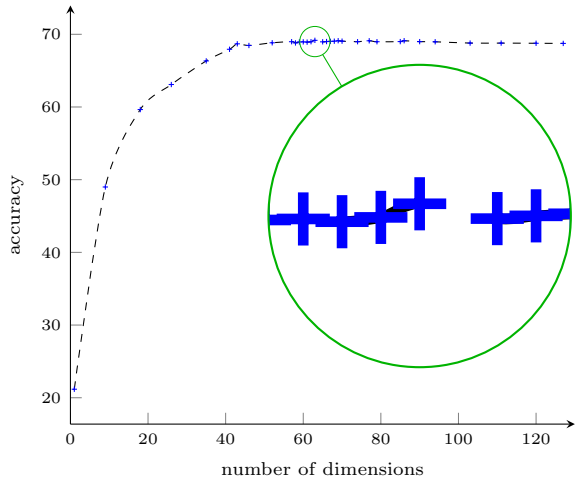


Figure 6: Accuracy of naive Bayes with respect to the number of PCA dimensions retained.

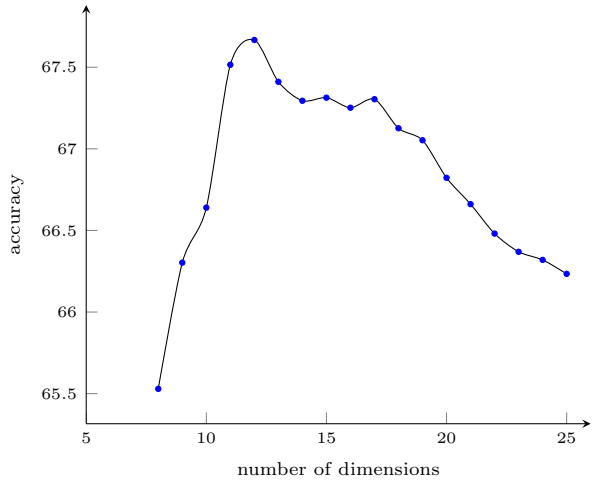


Figure 7: Accuracy of naive Bayes with respect to the number of LDA dimensions retained.

$-\mathcal{L}$ and its gradient:

$$-\frac{\partial \mathcal{L}}{\partial w_i} = -\sum_{t=1}^{N} \left(x_i^t (r^t - \text{sigmoid}(\mathbf{w}^T x)\right) + 2\lambda \mathbf{w}$$

Then I trained a classifier for each class against all the other. At the prediction step, each sample is affected a probability to belong to each class via the sigmoid function and the greatest one is selected.

I have done a 4-fold cross validation to find the parameter λ and it finally give an accuracy of about 76.3% for $\lambda = 10^{-2}$, see Figure 8. But it degrades when I do PCA before so I used the training data untouched. I have not had time to see if training $K(K+1)/2$ pairwise classifiers would give better results, but that sounds promising because it is more likely that two classes are linearly separable compared with one class against twenty five others. Again, this generalization accuracy was optimistic and I only get 69.8% accuracy on the test set.

Compared with naive Bayes' approach, logistic regression has the advantage of making less assumptions on the data. And obviously, its results are somewhat better. Yet, it suffers from longer training time (mainly because of the iterative optimization phase) and it is still a linear model.

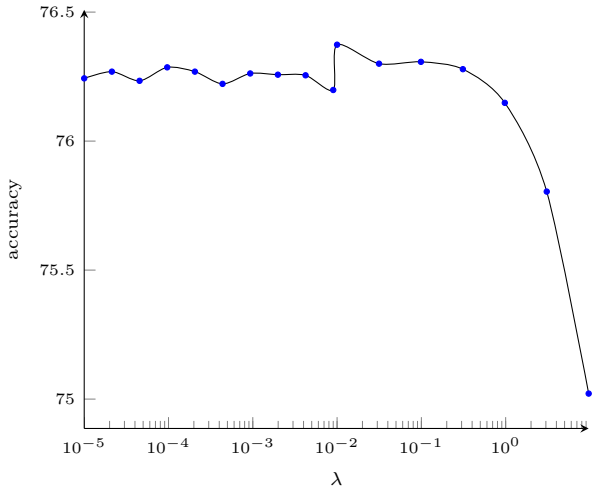


Figure 8: Accuracy of logistic regression with respect to λ.

5. SVM

Assume that we have two classes labeled by $+1$ and -1. The idea in SVM is to find $\mathbf{w}$ and w_0 such that for all points in our set $\mathcal{X} = \{r^t, x^t\}_{t=1\ldots N}$, $r^t(\mathbf{w}^T x^t + w_0) \geq 1$ holds. The distance to discriminant is

$$\frac{r^t(\mathbf{w}^T x^t + w_0)}{||\mathbf{w}||} \geq \rho$$

so we want to maximise ρ s.t. $\rho\,||\mathbf{w}|| = 1$ i.e. minimize $||\mathbf{w}||^2/2$. But to handle non linearly separable cases, we add a "soft error" $C\sum_{t=1}^N \xi^t$ so that the classification condition is now written as

$$r^t(\mathbf{w}^T x^t + w_0) \geq 1 - \xi^t$$

meaning that $\xi^t = 0$ for points far enough from the margin, $\xi^t \in]0, 1]$ for those which are well classified but lied in the margin and $\xi^t > 1$ for the misclassified ones.

So we write the problem in its Lagrangian form

$$L_p = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{t=1}^N \xi^t - \sum_{t=1}^N \mu^t \xi^t - \sum_{t=1}^N \alpha^t \left(r^t(\mathbf{w}^T x^t + w_0) - 1 + \xi^t\right)$$

and we want to minimize L_p w.r.t $\mathbf{w}, w_0$ and maximise it w.r.t α^t, s.t $\alpha^t \geq 0$ and $\mu^t > 0$.

Then we take the dual problem of maximising L_p w.r.t α^t s.t. $\nabla L_p = 0$ and $\alpha^t \geq 0$. Setting the derivative of L_p w.r.t $\mathbf{w}$ to 0 yields

$$\mathbf{w} = \sum_{t=1}^N \alpha^t r^t \alpha^t \tag{1}$$

and w.r.t w_0: $\sum_{t=1}^N \alpha^t r^t = 0$. Finally $\partial L_p/\partial \xi^t = C - \alpha^t - \mu^t = 0 \Leftrightarrow 0 \leq \alpha^t \leq C$. The problem is now to minimize[4]

$$\begin{aligned} L_d &= -\frac{1}{2}\mathbf{w}^t\mathbf{w} + \sum_{t=1}^N \alpha^t \\ &= -\frac{1}{2}\sum_{t=1}^N\sum_{s=1}^N \alpha^t\alpha^s r^t r^s (x^t)^T x^s + \sum_{t=1}^N \alpha^t \end{aligned}$$

s.t. $\sum_{t=1}^N \alpha^t r^t = 0$ and $0 \leq \alpha^t \leq C$

The few samples for which $\alpha^t > 0$ are called support vectors and define $\mathbf{w}$ according to (1). Those satisfying $r^t(\mathbf{w}^T x^t + w_0) = 1$ and $\alpha^t < C$ are on the border and defines w_0. C is a regularisation parameter representing a trade off between a minimal error on the training set and a maximum margin separation. If it is set too large, its high penalty will lead to a lot of support vector, meaning overffiting. Yet, if it is too small, the model is to simple and underfits the training set.

[4]Because other terms cancel out

The flexibility of SVM comes from the fact that we can replace the term $(x^t)^T x^s$ by a custom kernel $K(x, y)$ that should be large when x and y are "similar". One popular choice that I have adopted is the radial basis function

$$K(x, y) = \exp\frac{-||x-y||^2}{2\sigma^2}$$

which introduce a new parameter, σ^2.

By doing a cross-validated grid search to find σ^2 and C, I get $\sigma^2 = 10.8$ and $C = 5$ (see Figure 9, which give a 89.89% generalization accuracy. On the test set, it turns to be 89.50%. Nonetheless, these results are still much better than with the two previous methods. Another advantage of SVM is that they may be improved by using a more appropriate kernel to compare structured images than a simple euclidean distance. But they also come at the price of much longer training and testing times. And more importantly, the optimization problem have analytical solution but they are intractable due to the size of the $N \times N$ Gram matrix $\mathbf{K} = [K(x_i, x_j)]_{i,j}$, which holds more than 1.7 million elements in our case. So one have to rely on sophisticated numerical method described in theses papers (Joachims, 1999; Fan et al., 2005). And because I thought it would be too difficult, I download the libSVM package, which take only two lines of MATLAB but seven minutes of training on my laptop.

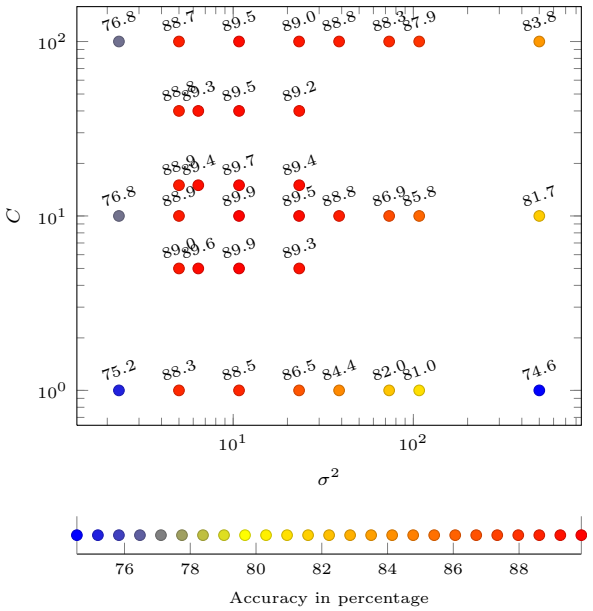


Figure 9: 3-fold cross-validated grid search.

6. Comments on the project

Overall, I found this project stimulating and it was interesting to see a real world application of the material presented in the course. My main difficulty was that during a long time in the beginning, I was a little bit lost at the number of available methods to tackle the problem, so I pondered a lot instead of trying one of them. When I started writing code, it feels better. I have not really kept track of the time spent on the project, but a rough estimate would be twenty hours. I could probably have been faster by spending less time writing this report, but it was a valuable LaTeX experience!

References

Alpaydin, E. *Introduction to machine learning*. MIT press, 2nd edition, 2009.

Fan, R.E., Chen, P.H., and Lin, C.J. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

Joachims, T. Making large-scale svm learning practical. In *Making large-Scale SVM Learning Practical*, chapter 11. MIT-Press, 1999.